UGANDA CHRISTIAN UNIVERSITY
A Centre of Excellence in the Heart of Africa

CSC1101: Structured Programming

Lecture 01 (BSCS, BSDS, BSIT)

Exception Handling with an OOP Focus

Mr. Ian Raymond Osolo
(Department of Computing & Technology
Faculty of Engineering, Design & Technology

Mon 16th October 2023

Exception Handling in Object-Oriented Python

Exception Handling with an OOP Focus

Lecture Objectives:

- Understand exceptions
- Handle errors effectively in OOP
- Create robust Python classes

Introduction to Exceptions and Errors

- Definition: What is an error? What is an exception?
- - Errors: stop program execution.
- - Exceptions: disrupt runtime flow.

Introduction to Exceptions and Errors

- Syntax Errors vs. Runtime Exceptions
- - Example: Incorrect syntax vs. division by zero.

```
if True
    print("Syntax error!")  # Missing colon
```

Common Exceptions

- Examples:
- - ZeroDivisionError
- - FileNotFoundError
- - KeyError, IndexError, TypeError

ZeroDivisionError

- Occurs when you try to divide a number by zero. This is mathematically undefined.
- 10 / 0 # Raises ZeroDivisionError
- Can use an if statement to check if the divisor is zero before performing the division.

FileNotFoundError

- Raised when you try to access a file that “doesn't exist”.
- Could result from:
  - incorrect file path.
  - deleted or moved file
  - insufficient permissions to access the file.
- Double-check the file path for typos.
- Use a try-except
- Consider using a file existence check (os.path.exists()) before attempting to open the file.

Others;

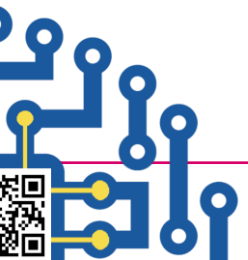
- KeyError in dicts. Can use the in operator (“key_name" in my_dict) to check if the key exists before accessing it.
- IndexError: when you try to access an index that is out of range.
- TypeError: Occurs when an operation or function is applied to an object of an inappropriate type.
  - try to add a string to an integer.
  - pass the wrong number of arguments to a function.
  - call a method on an object that doesn't support it.
  - Fix with type conversion

Basic Syntax of Exception Handling

try-except Structure:

```
try:
    # Code block that may raise an exception
except ExceptionType:
    # Code to execute if that exception occurs

finally:
    # Code that will run no matter what happens in try or except
```



Example: Handling Division by Zero

```
class Calculator:
    def divide(self, numerator, denominator):
        try:
            return numerator / denominator
        except ZeroDivisionError:
            return "Cannot divide by zero."

calc = Calculator()
print(calc.divide(10, 0))  # Output: "Cannot divide by zero."
```

Raising Exceptions

- Using **raise** to Enforce Rules
- raise ExceptionType("Error message")

raise ValueError("Invalid input")

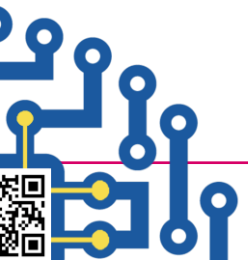
```
class InputValidator:
    def get_positive_number(self):
        number = int(input("Enter a positive number: "))
        if number < 0:
            raise ValueError("Number must be positive")
        return number

validator = InputValidator()
try:
    result = validator.get_positive_number()
    print(f"You entered: {result}")
except ValueError as e:
    print(f"Error: {e}")
```

Using else and finally

- The **else** Block:
  - - Runs only if no exception occurs.
- Useful when you want to execute some code only if no exceptions were raised.

```
try:
    number = int(input("Enter an integer: "))
except ValueError:
    print("That's not a valid integer.")
else:
    print(f"You entered {number} successfully.")
```



The finally Block: Executes regardless of whether an exception is raised.

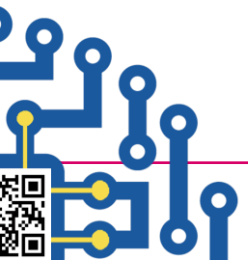
```
class FileManager:
    def read_file(self, filename):
        try:
            file = open(filename, 'r')
            return file.read()
        except FileNotFoundError:
            print(f"File '{filename}' not found.")
        finally:
            print("Closing the file.")
            if file:
                file.close()

manager = FileManager()
content = manager.read_file('data.txt')
```

Creating Custom Exceptions

- User-Defined Exceptions:
- - Why and how to create custom exceptions.
- class CustomException(Exception):

```
    def __init__(self, message="An error occurred"):
        self.message = message
        super().__init__(self.message)
```



```
class NegativeValueError(Exception):
    def __init__(self, message="Value cannot be negative"):
        self.message = message
        super().__init__(self.message)

class BankAccount:
    def __init__(self, balance):
        if balance < 0:
            raise NegativeValueError("Initial balance cannot be negative.")
        self.balance = balance

try:
    account = BankAccount(-100)
except NegativeValueError as e:
    print(f"Account creation error: {e}")
```

Exception Handling in OOP Context

- Integration in OOP:
- - Signal failures through raised exceptions.
- - Keep code clean and modular.

```
class BankAccount:
    def __init__(self, balance=0):
        self.balance = balance

    def deposit(self, amount):
        if amount <= 0:
            raise ValueError("Deposit amount must be positive.")
        self.balance += amount

    def withdraw(self, amount):
        if amount > self.balance:
            raise ValueError("Insufficient funds.")
        self.balance -= amount

try:
    account = BankAccount(100)
    account.deposit(-50)
except ValueError as e:
    print(f"Transaction error: {e}")
```

Best Practices for Exception Handling

- Avoiding Generic **except** Clauses:
- - Bad Practice Example: Catching all exceptions.

```
try:
    # Some code that might raise various exceptions
    x = 10 / 0  # ZeroDivisionError
    int("abc") # ValueError
except:
    print("An error occurred")
```

Good Practice: Catching specific exceptions.

```
try:
    # Some code that might raise various exceptions
    x = 10 / 0
    int("abc")
except ZeroDivisionError:
    print("Cannot divide by zero")
except ValueError:
    print("Invalid value for conversion to integer")
```

Assignment (Homework)

- Assignment Task: StudentManagement class with methods for adding, updating, and deleting students, **use exception handling** where needed.
- Just like the banking example (add it to previous assignment)

Uganda Christian University
P.O. Box 4 Mukono, Uganda
Tel: 256-312-350800
https://ucu.ac.ug/ Email: info@ucu.ac.ug.
@ugandachristianuniversity @UCUniversity
@UgandaChristianUniversity

Department of Computing & Technology
FACULTY OF ENGINEERING, DESIGN AND TECHNOLOGY
Tel: +256 (0) 312 350 863 | WhatsApp: +256 (0) 708 114 300
@ucucomputeng @ucu_ComputEng
https://cse.ucu.ac.ug/ Email: dct-info@ucu.ac.ug